A 3-D Lasso Tool for Editing 3-D Objects: Implemented Using a Haptics Device

Marjorie Darrah
Institute for Scientific Research
Fairmont, WV 26554
mdarrah@isr.us

Alicia Kime
School of Science and Mathematics
Fairmont State College
Fairmont, WV 26554
akime@mail.fscwv.edu

Frances Van Scoy
Lane Department of Computer Science and Electrical Engineering
West Virginia University
Morgantown, WV 26505
fvanscoy@wvu.edu

Abstract

Many paint programs include a lasso editing tool that allows the user to capture an irregular portion of a two-dimensional figure. However, no corresponding tool is known to exist for three-dimensional editing. This paper describes a lasso tool, based on a convex hull algorithm, that has been simulated using the SensAble™ PHANTOM™ haptic device. The PHANTOM™ device is employed as a three-dimensional mouse to select a set of non-planar voxels, that is, three-dimensional pixels. The algorithm uses the selected voxels to define a convex hull. Once the voxels within the convex hull have been identified, they can be deleted, copied, or modified.

1 Introduction

This project addressed the problem of selecting a set of voxels (three-dimensional pixels) in a three-dimensional model for editing. The tools available for two-dimensional editing are rather flexible; many paint programs allow the mouse to sketch the boundary of a region, either freehand or using line segments to form a polygon. The user does not have to enclose the region in a circle, ellipse or square but may select a shape that is irregular. We sought to generalize the two dimensional idea of a lasso enclosing a irregular polygon to a lasso for three-dimensional sets of voxels.

Current packages on the market allow three-dimensional data to be edited in several ways including what are known as: the cookie-cutter (or extrusion) approach, the melon baller approach and the two-dimensional slicing approach [1]. The cookie-cutter approach allows an arbitrary shape to be drawn on a plane to enclose an area, then extends the shape downward orthogonal to the plane so that it selects a volume much like a cookie cutter presses through a large quantity of dough. The melon baller method uses a regular shape, such as sphere, cube, or rectangular prism to carve out sections of the three-dimensional space by using that shape. This allows greater control over the volume selected than the previous method, but it still yields a regular shaped region. In a third approach, two-dimensional slicing, a two-dimensional projection of the three-dimensional data set is edited repeatedly. The data set may be rotated to observe the data from different viewpoints, but the mouse is still editing in two dimensions.

The tool we created and simulated using the PHANTOM™ haptics device makes possible the freehand editing of a three-dimensional data set. Our prototype allows the user to specify a data set, containing points with (x,y,z) coordinates, which is then rendered as voxels in a three-dimensional haptic scene. To construct a convex hull, the haptics device is first used to select a set of three-dimensional voxels. When at least four non-planar voxels have been chosen, the program will construct a convex hull that encloses all points within the boundary of an irregular convex polyhedron defined by those voxels.

2 Mathematical Approach

To create a lasso tool that allows for the arbitrary selection of points and the enclosure of irregular three-dimensional volumes, we first explored the geometry of the problem. Just as the two-dimensional lasso tool uses a polygon to enclose the area to be edited, a three-dimensional lasso must define a polyhedron. A polyhedron is a region of space whose boundary is composed of flat polygon faces, any pair of which are either disjoint or meeting at edges and vertices.

It should be noted that we here are discussing and constructing a *convex* polyhedron. A set $\mathbf{S}$ of points is defined as convex if $\mathbf{x} \in \mathbf{S}$ and $\mathbf{y} \in \mathbf{S}$ implies that the line segment $\mathbf{xy} \subseteq \mathbf{S}$. A polyhedron that is constructed from the user-selected points is called a *convex hull* of the set. The following example may help visualize the concept of a convex hull. In a two-dimensional plane, the convex hull is the shape a string assumes when anchored to a nail at the lowest point with respect to y and wrapped counterclockwise around nails pounded into the plane at each point. In three dimensions, the *boundary* of a convex hull is the shape taken by plastic wrap stretched tightly around all the points. More formally a definition of convex hull of a set **S** is the intersection of all convex sets that contain **S**.

Many algorithms exist for constructing both two and three-dimensional convex hulls. Graham's Algorithm [1], the fastest method for constructing a convex hull in two dimensions, has no obvious generalization to three dimensions. Among the best-known three-dimensional algorithms are Gift Wrapping (Chand & Kapur), Divide and Conquer (Preparata & Hong) and Incremental Algorithm (Seidel & Kallay). We chose to implement the Incremental Algorithm outlined by Joseph O'Rourke in the second edition of his, *Computational Geometry in C* [2].

In two dimensions the basic method of the Incremental Algorithm is to add points one at a time, constructing the hull of the first *k* points at each step and using that hull to incorporate the next point.

```
Algorithm: INCREMENTAL ALGORITHM
Let $H_2 \leftarrow \text{conv}\{p_0,p_1,p_2\}$.
For $k = 3$ to $n-1$ do
    $H_k \leftarrow \text{conv}\{H_{k-1} \cup p_k\}$
```

The algorithm begins with a triangle and considers whether the next point is inside or outside the area enclosed by that triangle. If the point is inside, then it is discarded; if it is outside, then the algorithm determines the two tangent lines from the new point to the triangle. The algorithm continues in this manner until the convex hull is fully defined.

This incremental algorithm, with time complexity of $O(n^2)$, can be generalized to three dimensions.

```
Algorithm: 3D INCREMENTAL ALGORITHM
Initialize $H_3$ to tetrahedron $(p_0,p_1,p_2,p_3)$.
for i = 4, …, $n-1$ do
for each face f of $H_{i-1}$ do
    Compute volume of tetrahedron determined by $f$ and $p_i$.
    Mark $f$ visible iff volume < 0.
    if no faces are visible
        then
            Discard $p_i$ (it is inside $H_{i-1}$).
        else
            for each visible border edge $e$ of $H_{i-1}$ do
                Construct cone face determined by $e$ and $p_i$.
            for each visible face $f$ do
                Delete $f$.
            Update $H_i$.
```

The overall structure of the three-dimensional incremental algorithm is identical to that of the two-dimensional version. From a set of identified points, the algorithm chooses four non-planar points from which the initial hull is constructed. At the *ith* iteration, the algorithm computes the hull-in-progress by adding a new point. This action yields two possibilities: (1) if the new point is inside the existing hull, it is discarded; (2) if it is outside, the algorithm constructs a cone face determined by the new point and each border edge visible from that point. The process of defining the convex hull on the basis of a single new point *p* is visualized in Figures 1 - 3.

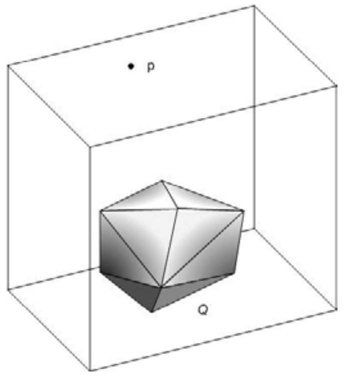

Figure 1: Convex hull *Q* and point *p* outside the hull

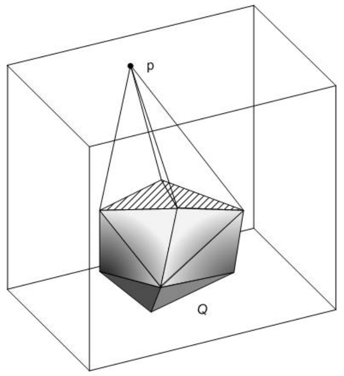

Figure 2: The striped faces are interior and marked for deletion when the hull is extended to include point *p*.

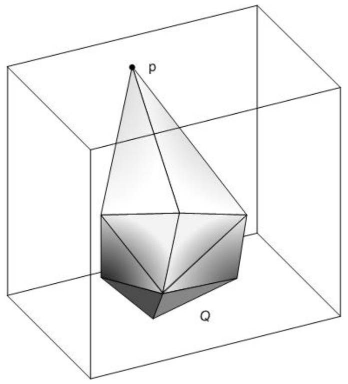

Figure 3: The hull after point p and the new faces have been added.

3 Implementation of Convex Hull Algorithm

To simplify the data structures used, the algorithm assumes that every face of the surface of the *polytope* (convex polyhedron) is a triangle. Three primary data types are required to define a convex hull, vertices, edges, and faces. The computer program that implements the algorithm consists of four basic sections: read, create initial polytope, construct the hull, and print. First the vertices are read into an array. Next, an initial polytope for the incremental algorithm is created. It is a double-sided triangle, a polyhedron with three vertices and two faces that are identical except for the order of their vertices. To construct this figure, three noncollinear points are found, and after marking them as processed the double-sided triangle is built as the first hull *Q*. It is now necessary to find a fourth nonplanar point *p* to form a tetrahedron. For each point *p* it must be determined if *p* is inside or outside *Q*. To accomplish this, one must determine for every face *f* of *Q* if the face *f* is *visible* from point *p*. The face *f* is visible from *p* iff it lies in the positive halfspace determined by the plane containing *f*. The positive side is determined by the counterclockwise orientation of *f*. If no face is visible from *p,* then it must lie inside the hull, and it is marked for deletion. If *p* is outside *Q*, then the hull is transformed by finding the tangent planes that bound a cone of triangle faces with the apex at point *p* and the base edges of *Q* (Figure 2). The portion of the polytope visible from *p* forms a connected region of the surface of the existing hull as indicated by the striped faces in Figure 2. The interior faces of the region must now be deleted and the cone connected to the boundary. To determine the edges of the hull, those edges adjacent to two visible faces are marked interior and marked for deletion; those edges with one adjacent visible face are identified as on the border of the visible region. Once the edges of the border are identified, then for every edge on the border a new triangular face can be constructed using that edge and the point *p*. At this point, the convex hull is almost complete. All that remains is to "clean up" the hull by deleting hidden faces and edges and making sure the vertex, edge and face lists are properly linked.

4 Construction of the Prototype

Our effort to model the algorithm began by building a visualization program using OpenGL. The program we devised first reads in a data set consisting of (x,y,z,r,g,b) giving position and color information and then creates a voxel for each point in an *n* x *n* x *n* space. The voxels are cubes that can be opaque or transparent. Once the scene has been rendered, it can be rotated clockwise or counterclockwise in the x, y, and z directions around a fixed point. Given a set of three-dimensional non-planar coordinates (a subset of the original data set) the program determines the convex hull defined by those points. In our program we identified the hull by changing the color of the points that constituted it. When operational the program demonstrated that the convex hull algorithm could be implemented in a graphics program.

Wishing additionally to implement the model into a three-dimensional touch environment, we developed a related program using the GHOST® API to incorporate haptics. The program we developed for use on the PHANTOM™ haptics device uses "voxel world" coordinates, where the position of the voxel is (x, y, z) and x, y, and z are all integers. As a trial experiment, we created a 3 x 3 x 3 grid of spheres to represent the voxel space. The spheres were large enough to be felt, but separated by sufficient space to permit movement of the haptic cursor within the grid. We made use of the PHANTOM™ haptics device as a three-dimensional selection device. Using the stylus, we selected certain spheres that represented the set of points to define our convex hull. The algorithm then determined the interior points of the hull and we identified those voxels by changing their color.

5 Future Work

One application of a tool such as we devised is to facilitate the exploration of three-dimensional scientific or abstract data sets using haptics. At the present we hope to incorporate this work into a tool kit that uses haptic feedback to explore Light Detection and Ranging (LIDAR) data. The LIDAR tool kit will enable the user to be immersed in and interact with a point cloud of data and the 3-D lasso editing tool will allow for selection of points for removal or coloring.

References

[1] Van Scoy, F., Peredera, A., and Kime, A. (1999), "A 3-D Lasso Tool for Editing 3-D Objects: Preliminary Work", Workshop on Virtual Reality, Marilia, Brazil, November 18-20.

[2] O'Rourke, J. (1998), *Computational Geometry in C*, second edition, Cambridge University Press, Cambridge.

[3] Graham, R.L. (1972), "An efficient algorithm for determining the convex hull of a finite planar set", *Inform. Process. Lett.* **1**, 132-3.